9 Working with Array and String

In chapter 7, we studied basic data types supported in Java. Each variable of such basic data type can store only one value at a time. Such variables are called scalar variable. In this chapter, we will discuss some composite data types that can be used to store a collection of more than one data values. For example, array and string.

Introduction to Array

An array is a variable representing a collection of homogeneous type of elements. Arrays are useful to represent vector, matrix and other multi-dimensional data. Vector is a one dimensional (1-D) data structure that can be used to store list of items like characters, numbers. Matrix is used to represent two dimensional (2-D) data structure like table of rows and columns.

Arrays are useful when same operations are to be performed on various elements of the similar type. All the elements of an array are stored in memory using contiguous storage space. Each element is identified by an index position associated with array variable.

In Java, array is an object used to manage list of items. Creating an array is a two step process:

1. Declare an array object
2. Create an array object

An array object can be created in two ways:

1. Using new operator and specifying the size
2. Directly initializing the content of array

1-D array

Array with single dimension is known as 1-D array. For example, vector which can be seen as a collection of one or more scalar variables. Instead of declaring individual variables like marks1, marks2, marks3, marks4, mark5; we can declare array marks[5] and accesses individual variable elements as marks[0], marks[1], marks[2], marks[3], marks[4].

To declare a 1-D array we use a pair of square brackets [ ] after array name or after data type. The syntax to declare array is as follows:

```
<data type> <array name> [];    or <data type> [] <array name>;
```

For example, to store the marks obtained by a student in five tests in Mathematics subject, we can use an array of five integer elements. Array object named 'marks' with size for 5 elements can be created as follows:

```
int marks[];                    // declare array object
marks = new int [5];            // create array object
```

We can combine above two steps and create array using a single statement also as follows :

```
int marks[] = new int [5];     or     int [] marks = new int [5];
```

Here name of the array is 'marks'. It refers to memory location where five integer values are stored. Integer of int data type uses 4 bytes storage space. Thus, array 'marks' requires 5 x 4 = 20 bytes in contiguous locations in memory.

To refer an element of an array, we use index or subscript in square bracket [ ] after array variable name as shown in figure 9.1. Index specifies the position of an element in an array; for example, marks[2]. Index value starts from 0.

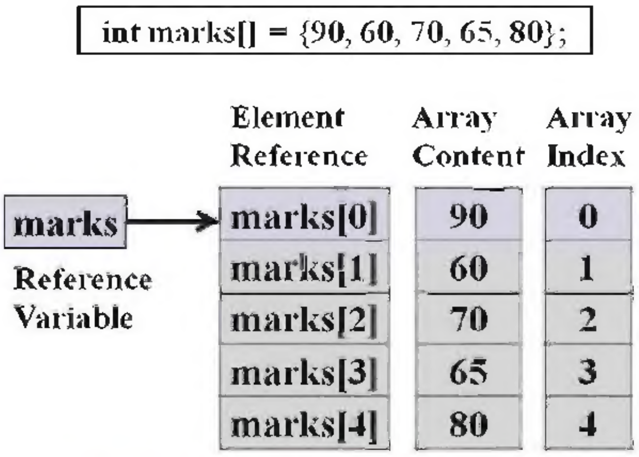
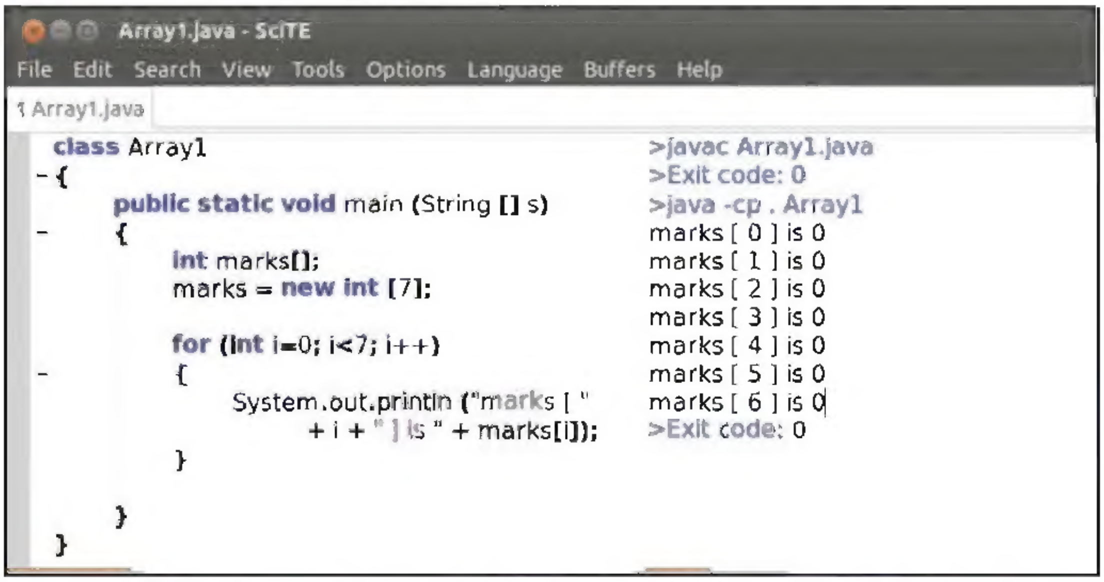


Figure 9.1 : One dimensional array

In figure 9.1 the array variable marks has an index value from 0 to 4. Here, marks[0] refers to the first element and marks[4] refers to the last element.

As explained in chapter 8, an object is a reference variable. Array being an object in Java, array name is also a reference variable. It contains reference to memory location where the array elements are stored. See figure 9.1.

As array is an object, so its elements are initialized with default values. Figure 9.2 shows an example of how to use an array in java programs.



```
class Array1
{
    public static void main (String [] s)
    {
        int marks[];
        marks = new int [7];

        for (int i=0; i<7; i++)
        {
            System.out.println ("marks [ "
                    + i + " ] is " + marks[i]);
        }

    }
}
```

```
>javac Array1.java
>Exit code: 0
>java -cp . Array1
marks [ 0 ] is 0
marks [ 1 ] is 0
marks [ 2 ] is 0
marks [ 3 ] is 0
marks [ 4 ] is 0
marks [ 5 ] is 0
marks [ 6 ] is 0
>Exit code: 0
```

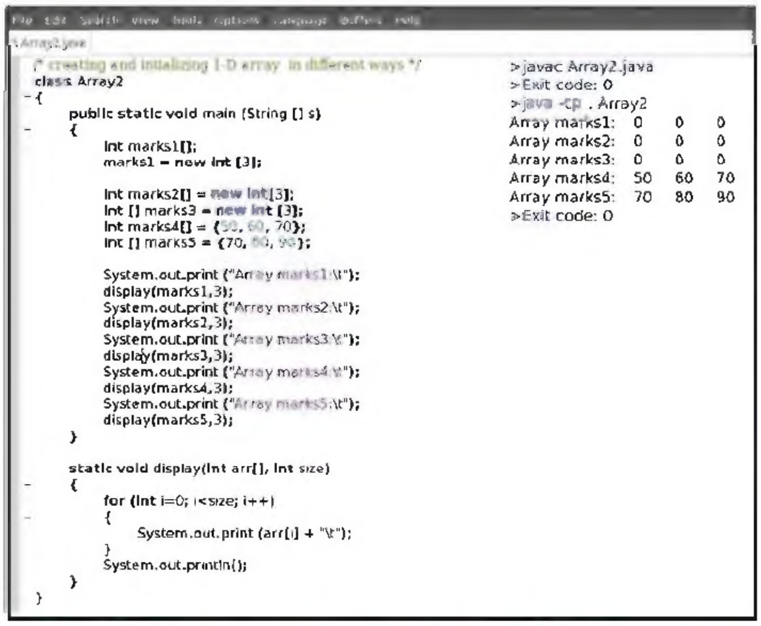
Figure 9.2 : Array object with its elements initialized by default values

One can also specify the initial values of data elements while declaring an array. 1-D array is initialized using comma separated values of data elements in braces { }. For example,

int marks[ ] = {90, 70, 77}; or int [] marks = {90, 70, 77};

Note that when an array is initialized while creating, it does not need the use of new operator. Its size is same as the number of values specified in braces.

The code shown in figure 9.3 uses different ways to create and initialize array. Note the use of class method display() used to display contents of an array. Class methods are declared as static and can be referred without any object in the class.

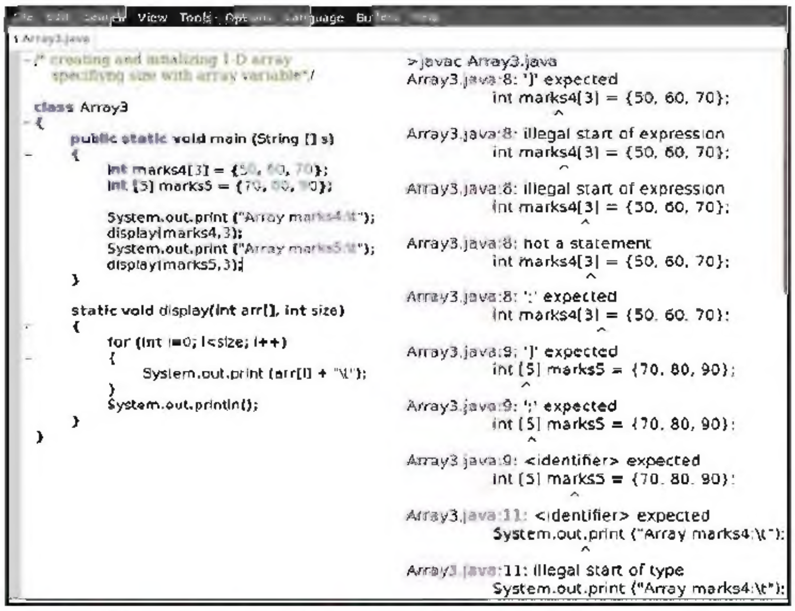


```
/* creating and initializing 1-D array in different ways */
class Array2
{
    public static void main (String [] s)
    {
        int marks1[];
        marks1 = new int [3];

        int marks2[] = new int[3];
        int [] marks3 = new int [3];
        int marks4[] = {50, 60, 70};
        int [] marks5 = {70, 80, 90};

        System.out.print ("Array marks1:\t");
        display(marks1,3);
        System.out.print ("Array marks2:\t");
        display(marks2,3);
        System.out.print ("Array marks3:\t");
        display(marks3,3);
        System.out.print ("Array marks4:\t");
        display(marks4,3);
        System.out.print ("Array marks5:\t");
        display(marks5,3);
    }

    static void display(int arr[], int size)
    {
        for (int i=0; i<size; i++)
        {
            System.out.print (arr[i] + "\t");
        }
        System.out.println();
    }
}
```

```
>javac Array2.java
>Exit code: 0
>java -cp . Array2
Array marks1:  0    0    0
Array marks2:  0    0    0
Array marks3:  0    0    0
Array marks4:  50   60   70
Array marks5:  70   80   90
>Exit code: 0
```

Figure 9.3 : Different ways to create and initialize array object

In Java, we can not specify both the size of dimensions and initial values of the array elements simultaneously while declaring an array. Figure 9.4 shows the code to illustrate the statement. If we declare an array variable without initialization, only a variable is created to hold the reference as shown in figure 9.1. It does not create an array object, i.e. no memory is allocated for array elements. If we try to access the elements of an array, it will result in an error.

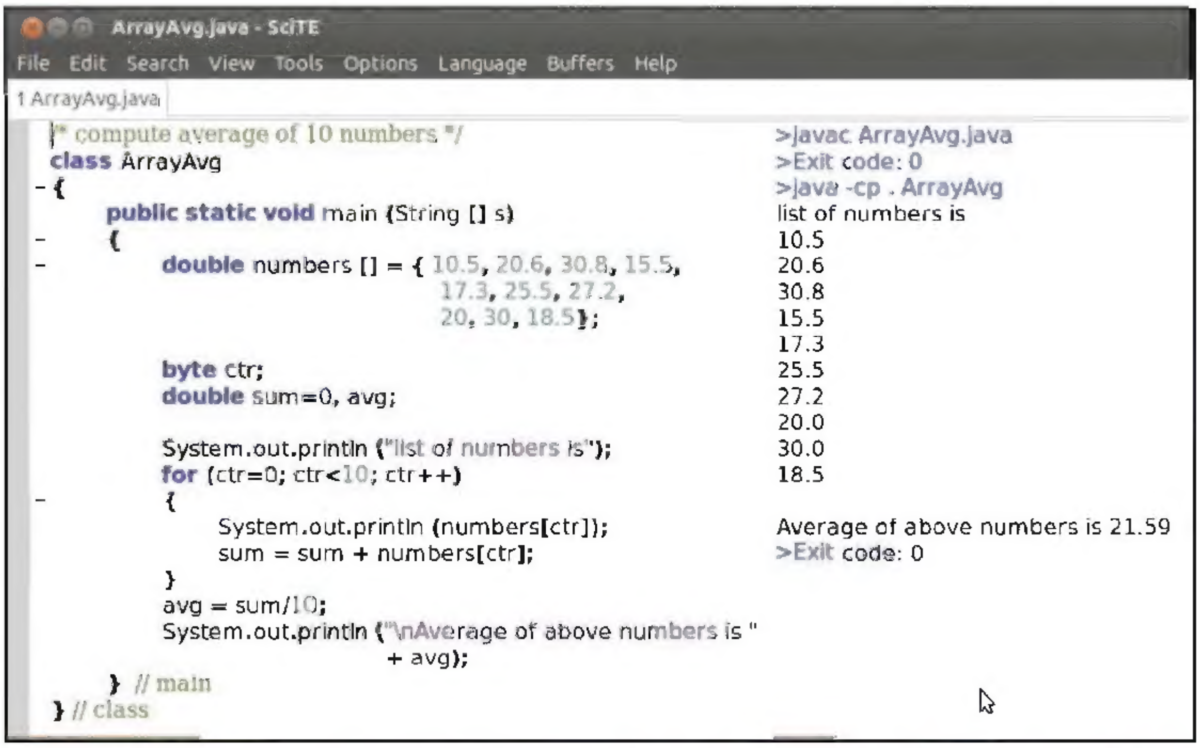


```
/* creating and initializing 1-D array
   specifying size with array variable*/

class Array3
{
    public static void main (String [] s)
    {
        int marks4[3] = {50, 60, 70};
        int [5] marks5 = {70, 80, 90};

        System.out.print ("Array marks4:\t");
        display(marks4,3);
        System.out.print ("Array marks5:\t");
        display(marks5,3);
    }

    static void display(int arr[], int size)
    {
        for (int i=0; i<size; i++)
        {
            System.out.print (arr[i] + "\t");
        }
        System.out.println();
    }
}
```

```
>javac Array3.java
Array3.java:8: ']' expected
            int marks4[3] = {50, 60, 70};
                       ^
Array3.java:8: illegal start of expression
            int marks4[3] = {50, 60, 70};
                       ^
Array3.java:8: illegal start of expression
            int marks4[3] = {50, 60, 70};
                         ^
Array3.java:8: not a statement
            int marks4[3] = {50, 60, 70};
                         ^
Array3.java:8: ';' expected
            int marks4[3] = {50, 60, 70};
                          ^
Array3.java:9: ']' expected
            int [5] marks5 = {70, 80, 90};
                 ^
Array3.java:9: ';' expected
            int [5] marks5 = {70, 80, 90};
                 ^
Array3.java:9: <identifier> expected
            int [5] marks5 = {70, 80, 90};
                       ^
Array3.java:11: <identifier> expected
            System.out.print ("Array marks4:\t");
                            ^
Array3.java:11: illegal start of type
            System.out.print ("Array marks4:\t");
```

Figure 9.4 : Illegal to specify size while initializing an array object

Every element of an array is an individual variable that can be referred by using index. Like variables, to change the value of an element, we can use assignment statement referring a variable element on left side. For example, marks[3] = 56; Let us write a program that computes an average of 10 floating point numbers and execute it. Observe the code listing and its output as given in figure 9.5.



```
/* compute average of 10 numbers */
class ArrayAvg
{
    public static void main (String [] s)
    {
        double numbers [] = { 10.5, 20.6, 30.8, 15.5,
                              17.3, 25.5, 27.2,
                              20, 30, 18.5};

        byte ctr;
        double sum=0, avg;

        System.out.println ("list of numbers is");
        for (ctr=0; ctr<10; ctr++)
        {
            System.out.println (numbers[ctr]);
            sum = sum + numbers[ctr];
        }
        avg = sum/10;
        System.out.println ("\nAverage of above numbers is "
                            + avg);
    } // main
} // class
```

```
>javac ArrayAvg.java
>Exit code: 0
>java -cp . ArrayAvg
list of numbers is
10.5
20.6
30.8
15.5
17.3
25.5
27.2
20.0
30.0
18.5

Average of above numbers is 21.59
>Exit code: 0
```

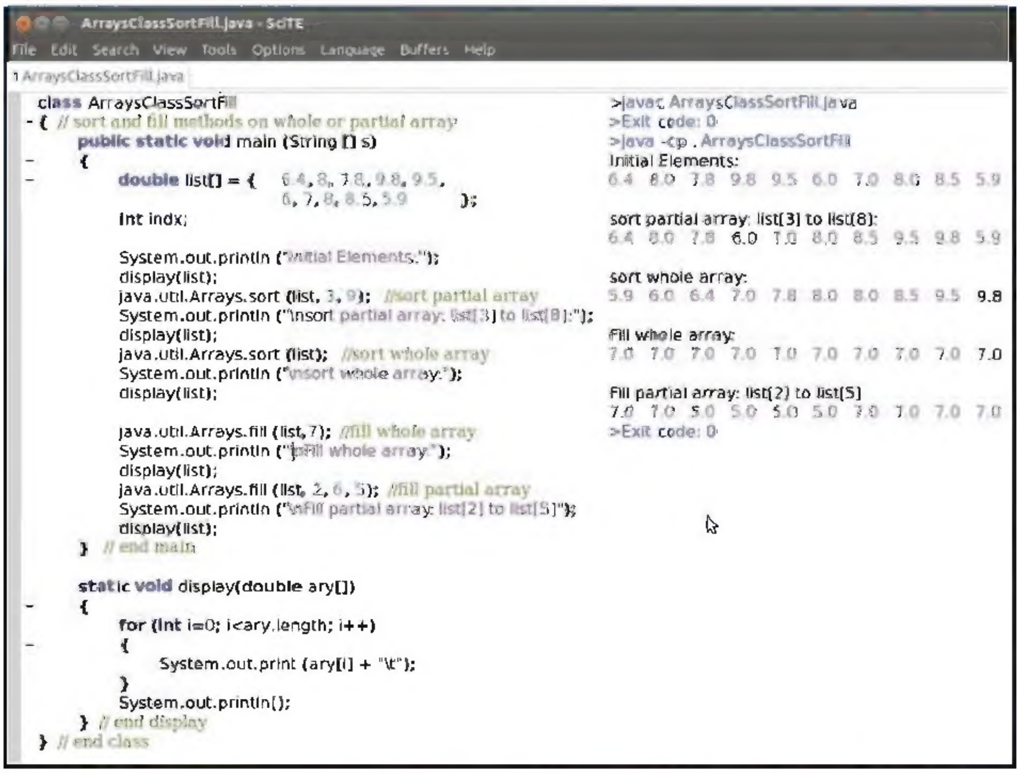
Figure 9.5 : Program to compute average of 10 numbers using 1-D array and loop

Here, we need to use the same operation 'add number to sum' repeatedly for different elements of same type. Use of an array helps in declaring 10 variables at a time and using a loop to perform same operation on different variable elements of an array.

We can perform various other operations on array. For example, compare two arrays, copy all the elements of one array to another, search for a specified element in array, sort elements of array and so on. For all such operations, we may write procedures as we did for finding average of elements. Instead of writing such code ourselves, we can use various static methods provided by Java using java.util.Arrays class. In Java, array is treated as an object of this Arrays class. This enables us to make use of methods of Arrays class with array type of data. The code listing and its output in figure 9.6 shows how to use methods sort() and fill() of java.util.Arrays class.

We can use sort() method to sort entire or part of array. When we use only array as an argument, it sorts an entire array. When we invoke this method with 3 arguments: array, start, last; it sorts partial array from element at index start to element at index (last-1). For example, the method call java.util.Arrays.sort (list, 1, 5) sorts elements of array list from list[1] to list[5-1]; See figure 9.6.

Method 'fill' is used to fill the whole or partial array with specified value. When the method is invoked with two arguments: array and value; it assigns the specified value to all array elements. When it is invoked with four arguments: array, start, last, value; it fills partial array from element at start to (last-1) with specified value. For example, fill (list, 7) assigns value 7 to all elements of list array; whereas fill(list, 2, 6, 5) assigns value 5 to elements list[2] to list[6-1]. See figure 9.6.



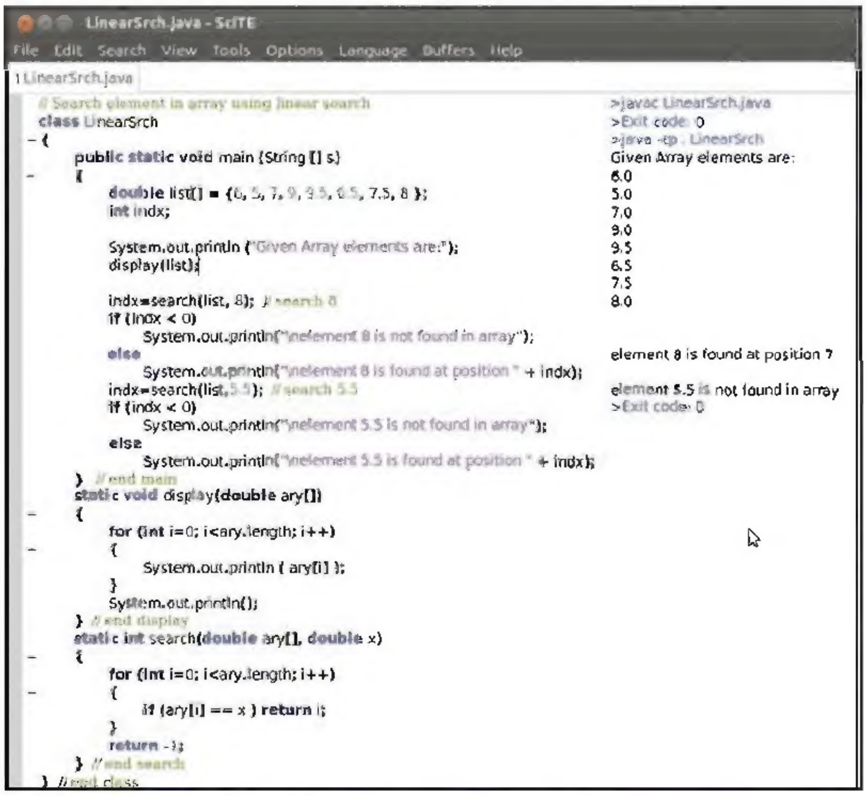
ArraysClassSortFill.java - SciTE
File Edit Search View Tools Options Language Buffers Help
1 ArraysClassSortFill.java

```java
class ArraysClassSortFill
{ // sort and fill methods on whole or partial array
    public static void main (String [] s)
    {
        double list[] = {    6.4, 8, 7.8, 9.8, 9.5,
                             6, 7, 8, 8.5, 5.9       };
        int indx;

        System.out.println ("Initial Elements:");
        display(list);
        java.util.Arrays.sort (list, 3, 9);  //sort partial array
        System.out.println ("\nsort partial array: list[3] to list[8]:");
        display(list);
        java.util.Arrays.sort (list);  //sort whole array
        System.out.println ("\nsort whole array:");
        display(list);

        java.util.Arrays.fill (list,7);  //fill whole array
        System.out.println ("\nFill whole array:");
        display(list);
        java.util.Arrays.fill (list, 2, 6, 5);  //fill partial array
        System.out.println ("\nFill partial array: list[2] to list[5]");
        display(list);
    }  // end main

    static void display(double ary[])
    {
        for (int i=0; i<ary.length; i++)
        {
            System.out.print (ary[i] + "\t");
        }
        System.out.println();
    }  // end display
} // end class
```

```
>javac ArraysClassSortFill.java
>Exit code: 0
>java -cp . ArraysClassSortFill
Initial Elements:
6.4  8.0  7.8  9.8  9.5  6.0  7.0  8.0  8.5  5.9

sort partial array: list[3] to list[8]:
6.4  8.0  7.8  6.0  7.0  8.0  8.5  9.5  9.8  5.9

sort whole array:
5.9  6.0  6.4  7.0  7.8  8.0  8.0  8.5  9.5  9.8

Fill whole array:
7.0  7.0  7.0  7.0  7.0  7.0  7.0  7.0  7.0  7.0

Fill partial array: list[2] to list[5]
7.0  7.0  5.0  5.0  5.0  5.0  7.0  7.0  7.0  7.0
>Exit code: 0
```

Figure 9.6 : Using sort() and fill() methods of Arrays class

To search an element in an array, Arrays class provides binarySearch() method. We will write our own method to search an element in array using linear search. Linear search method does element by element comparison in a serial fashion. Refer code listing and its output given in figure 9.7. It returns index position when an element is found; otherwise it returns -1.



```java
// Search element in array using linear search
class LinearSrch
{
    public static void main (String [] s)
    {
        double list[] = {6, 5, 7, 9, 9.5, 6.5, 7.5, 8 };
        int indx;

        System.out.println ("Given Array elements are:");
        display(list);

        indx=search(list, 8);  // search 8
        if (indx < 0)
            System.out.println("\nelement 8 is not found in array");
        else
            System.out.println("\nelement 8 is found at position " + indx);
        indx=search(list,5.5);  // search 5.5
        if (indx < 0)
            System.out.println("\nelement 5.5 is not found in array");
        else
            System.out.println("\nelement 5.5 is found at position " + indx);
    }  // end main
    static void display(double ary[])
    {
        for (int i=0; i<ary.length; i++)
        {
            System.out.println ( ary[i] );
        }
        System.out.println();
    }  // end display
    static int search(double ary[], double x)
    {
        for (int i=0; i<ary.length; i++)
        {
            if (ary[i] == x ) return i;
        }
        return -1;
    }  //end search
} //end class
```

```
>javac LinearSrch.java
>Exit code: 0
>java -cp . LinearSrch
Given Array elements are:
6.0
5.0
7.0
9.0
9.5
6.5
7.5
8.0

element 8 is found at position 7

element 5.5 is not found in array
>Exit code: 0
```

Figure 9.7 : Search an element in an array

2-D array

Two dimensional (2-D) arrays are used to store tabular data in the form of rows and columns. For example, to store 5 student's marks in 3 tests, we may use a tabular arrangement of marks in 5 rows and 3 columns as given in figure 9.8. In mathematics, we call it as a matrix of 5 rows and 3 columns where each element contains marks.

Students	Test1	Test2	Test3
1	50	60	70
2	35	30	50
3	70	75	80
4	80	85	90
5	40	50	55

Figure 9.8 : Tabular representation of 2-D array

In Java, 2-D array can be declared using array name and two pairs of square brackets [ ] [ ] to specify the size of two dimensions row and column respectively. For example, following statement declares and creates an array named marks to store 5 x 3 = 15 integer values in contiguous memory locations.

```
int marks [][] = new int [5][3];
```

Here, the logical view of array elements is a table of 5 rows and 3 columns. Physically, they are stored in memory using contiguous memory locations for 15 integers that is 60 bytes.

Java does not support multi-dimensional arrays directly. To create 2-D array, we have to create an array of array. There is no limit on number of dimensions.

In marks [5][3], it creates 1-D array object of 5 elements and each element is 1-D array object of 3 integers as shown in figure 9.9.

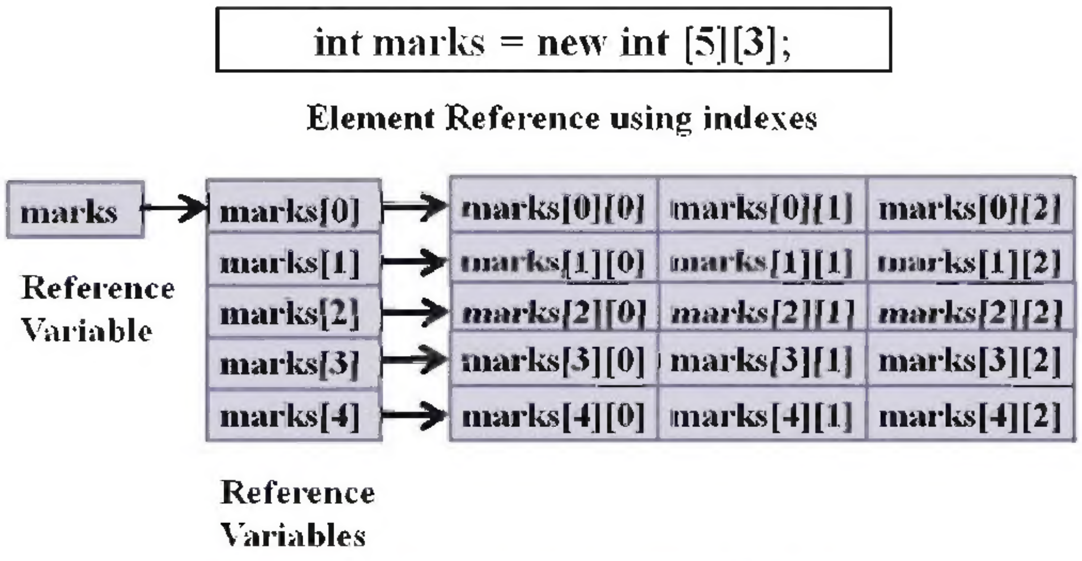
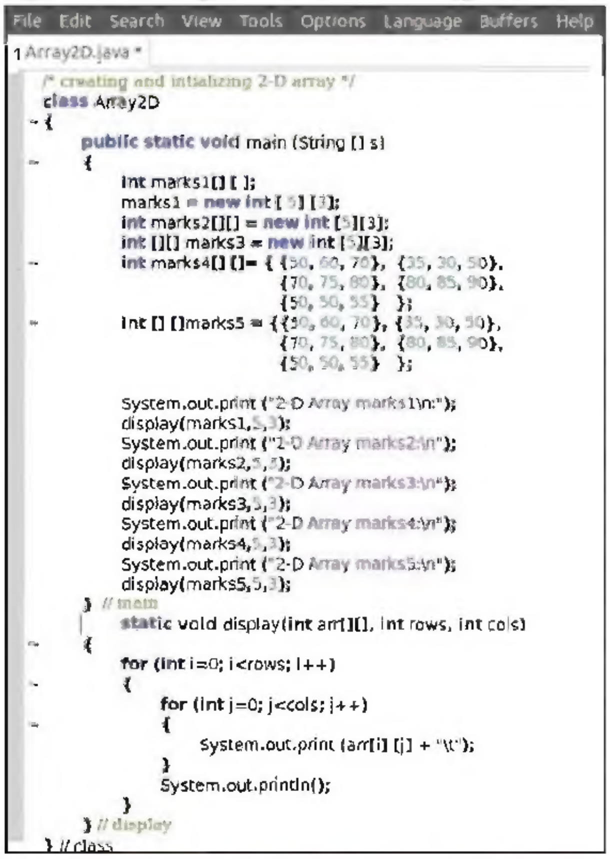


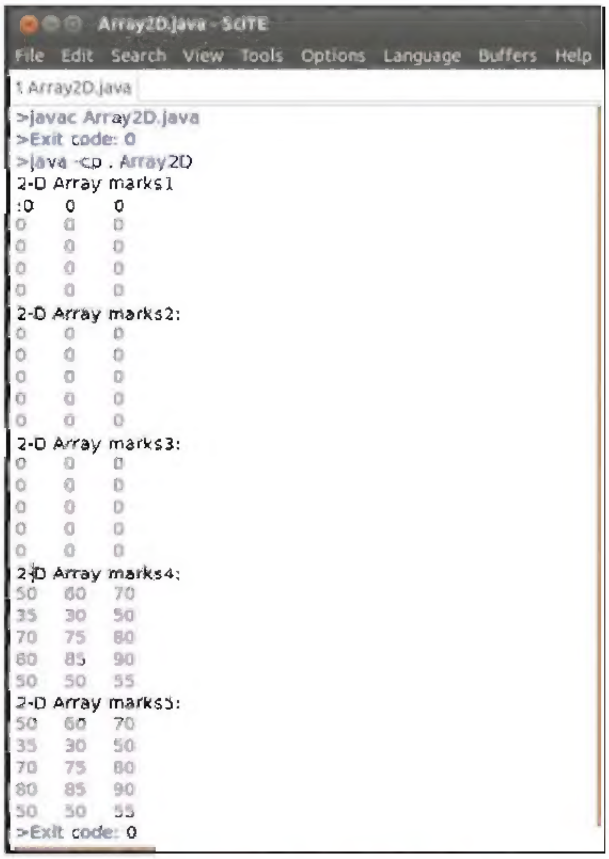
Figure 9.9 : 2-D array as an array of 1-D array

Declaration and initialization of 2-D array is very similar to 1-D array except the number of dimensions. In 2-D array, each row is considered as 1-D array element. Thus, like 1-D array initialization, each row is initialized by enclosing its elements in a pair of braces { } separated by comma. To initialize 2-D array, all these initialized rows are to be enclosed in curly braces {} separating them by comma (,). Like 1-D array, 2-D arrays can be declared in various ways as seen in code listing given in figure 9.10. Figure 9.11 shows output of the code.



```
/* creating and intializing 2-D array */
class Array2D
{
        public static void main (String [] s)
        {
                int marks1[] [ ];
                marks1 = new int [ 5] [3];
                int marks2[][] = new int [5][3];
                int [][] marks3 = new int [5][3];
                int marks4[] []= { {50, 60, 70}, {35, 30, 50},
                                   {70, 75, 80}, {80, 85, 90},
                                   {50, 50, 55}  };
                int [] []marks5 = {{50, 60, 70}, {35, 30, 50},
                                   {70, 75, 80}, {80, 85, 90},
                                   {50, 50, 55}  };

                System.out.print ("2-D Array marks1\n:");
                display(marks1,5,3);
                System.out.print ("1-D Array marks2:\n");
                display(marks2,5,3);
                System.out.print ("2-D Array marks3:\n");
                display(marks3,5,3);
                System.out.print ("2-D Array marks4:\n");
                display(marks4,5,3);
                System.out.print ("2-D Array marks5:\n");
                display(marks5,5,3);
        } // main
                static void display(int arr[][], int rows, int cols)
        {
                for (int i=0; i<rows; i++)
                {
                        for (int j=0; j<cols; j++)
                        {
                                System.out.print (arr[i] [j] + "\t");
                        }
                        System.out.println();
                }
        } // display
} // class
```

Figure 9.10 : Example of 2-D array



```
>javac Array2D.java
>Exit code: 0
>java -cp . Array2D
2-D Array marks1
:0     0     0
0      0     0
0      0     0
0      0     0
0      0     0
2-D Array marks2:
0      0     0
0      0     0
0      0     0
0      0     0
0      0     0
2-D Array marks3:
0      0     0
0      0     0
0      0     0
0      0     0
0      0     0
2-D Array marks4:
50     60    70
35     30    50
70     75    80
80     85    90
50     50    55
2-D Array marks5:
50     60    70
35     30    50
70     75    80
80     85    90
50     50    55
>Exit code: 0
```

Figure 9.11: Output

In Java, 2-D array is considered as an array of 1-D array. So, rows of 2-D array can have variable number of columns. Figure 9.12 shows how to use 2-D array of characters to store five names of computer programming languages.

```
char names [ ][ ] = {   {'J', 'a', 'v', 'a'}, {'C'}, {'C', '+', '+'},
          {'B', 'a', 's', 'i', 'c'},  {'P', 'a', 's', 'c', 'a', 'l'}    }
```

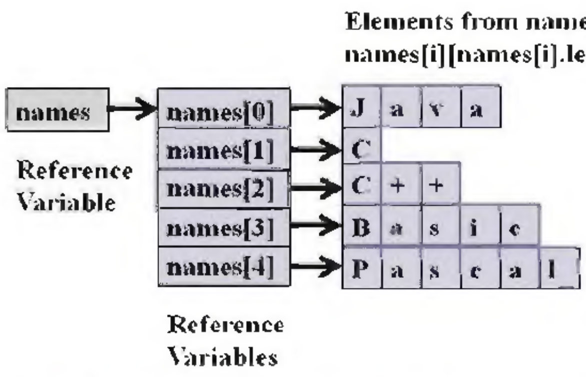
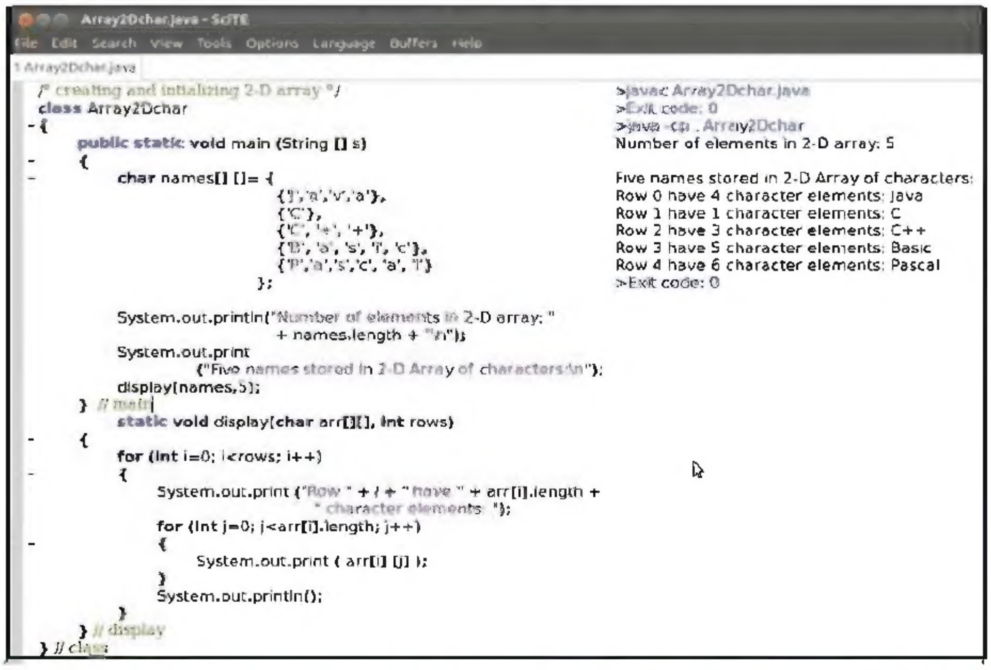


Figure 9.12 : 2-D array with variable number of columns

Each name is stored in different row and all names have different number of characters. Thus, each row is of different size. Size of each row can be known using 'length' property of 1-D array.

Figure 9.13 shows how to use 2-D array with varying column size. Here, we have used 'length' property to get the number of elements in 1-D array. For 2-D array, it returns number of rows. For 1-D array, it returns number of columns in specified row element. Remember that for 2-D array, number of elements is considered as number of rows and each row is 1-D array. In short, length property used with only array name returns the size of its first dimension.

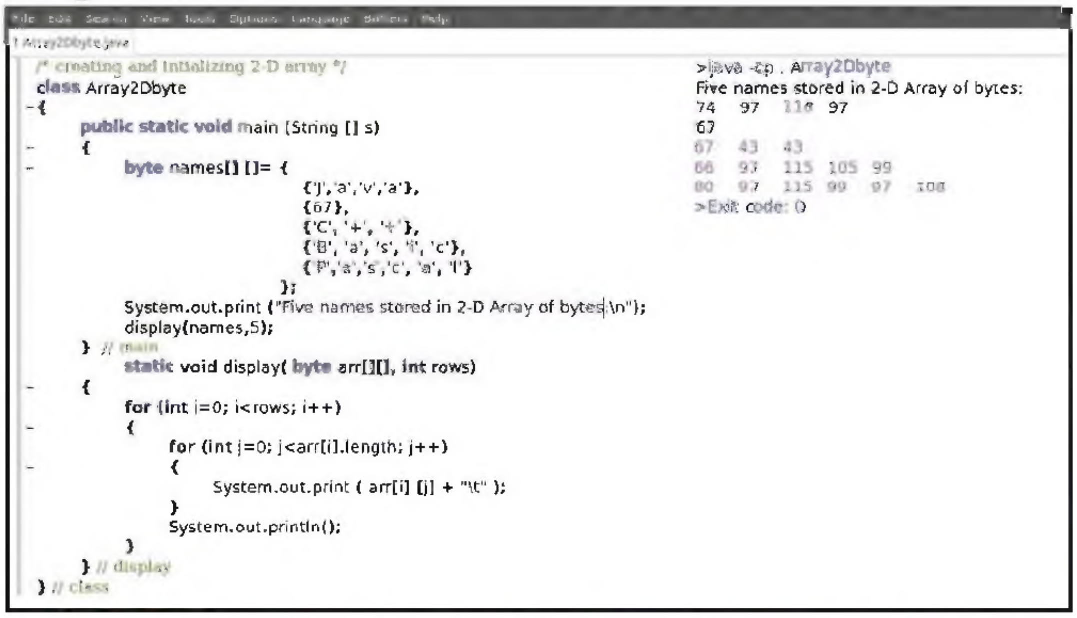


```
/* creating and intializing 2-D array */
class Array2Dchar
{
    public static void main (String [] s)
    {
        char names[] []= {
                            {'J','a','v','a'},
                            {'C'},
                            {'C', '+', '+'},
                            {'B', 'a', 's', 'i', 'c'},
                            {'P','a','s','c', 'a', 'l'}
                         };

        System.out.println("Number of elements in 2-D array: "
                            + names.length + "\n");
        System.out.print
                 ("Five names stored in 2-D Array of characters:\n");
        display(names,5);
    } // main
        static void display(char arr[][], int rows)
    {
        for (int i=0; i<rows; i++)
        {
            System.out.print ("Row " + i + " have " + arr[i].length +
                               " character elements: ");
            for (int j=0; j<arr[i].length; j++)
            {
                System.out.print ( arr[i] [j] );
            }
            System.out.println();
        }
    } // display
} // class
```

```
>javac Array2Dchar.java
>Exit code: 0
>java -cp . Array2Dchar
Number of elements in 2-D array: 5

Five names stored in 2-D Array of characters:
Row 0 have 4 character elements: Java
Row 1 have 1 character elements: C
Row 2 have 3 character elements: C++
Row 3 have 5 character elements: Basic
Row 4 have 6 character elements: Pascal
>Exit code: 0
```

Figure 9.13 : Program using 2-D array with variable number of columns

Later we will see that 1-D array of characters can be defined using String class available in Java. In figure 9.14, we have used 2-D array of bytes to store names. Here, it shows that initial chara type data values are converted to byte data type and the characters of names are assigned their corresponding ASCII values.



```
/* creating and intializing 2-D array */
class Array2Dbyte
{
    public static void main (String [] s)
    {
        byte names[] []= {
                            {'J','a','v','a'},
                            {67},
                            {'C', '+', '+'},
                            {'B', 'a', 's', 'i', 'c'},
                            {'P','a','s','c', 'a', 'l'}
                         };
        System.out.print ("Five names stored in 2-D Array of bytes:\n");
        display(names,5);
    } // main
        static void display( byte arr[][], int rows)
    {
        for (int i=0; i<rows; i++)
        {
            for (int j=0; j<arr[i].length; j++)
            {
                System.out.print ( arr[i] [j] + "\t" );
            }
            System.out.println();
        }
    } // display
} // class
```

```
>java -cp . Array2Dbyte
Five names stored in 2-D Array of bytes:
74    97    118   97
67
67    43    43
66    97    115   105   99
80    97    115   99    97    108
>Exit code: 0
```

Figure 9.14 : 2-D array: Characters stored in bytes using corresponding integer values

Points to be remembered :

- array is a collection of homogeneous type of data
- array elements can be accessed using index for each dimension in [ ]
- index value starts with zero
- multi-dimensional arrays can have variable size of 2nd, 3rd ... dimensions
- attribute 'length' is used to determine the size of the dimension
- array is treated as an object
- declaring an array without initialization does not create an array object

Strings

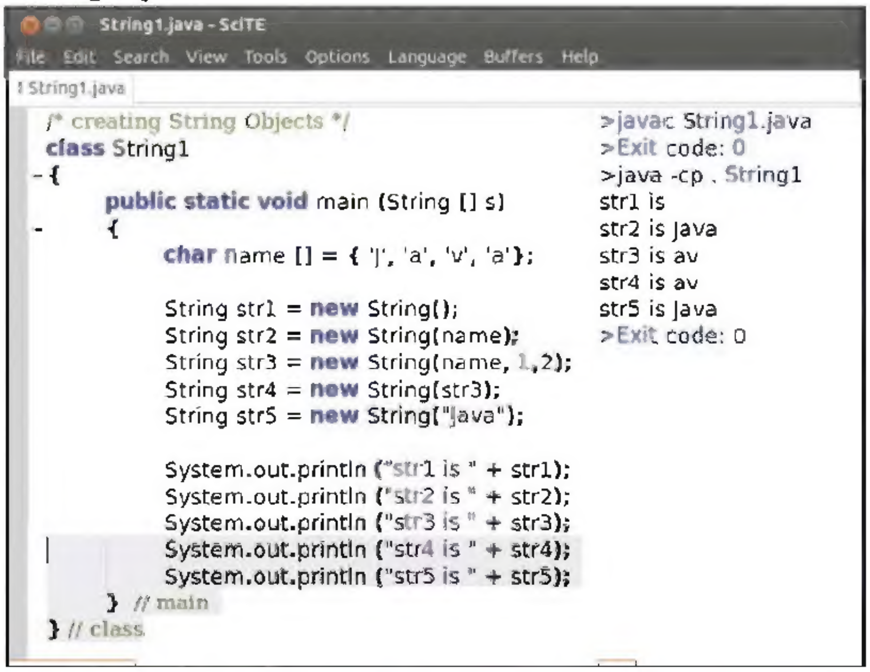
String is nothing but a sequence of characters. So, 1-D array of characters can be considered as a string. We have already studied and used string literals, where a sequence of characters is enclosed between double quotes.

To use variables that can store strings, Java supports two types of strings that are handled by two classes namely 'String' and 'StringBuffer'. In this chapter, we will see the first one.

Some of the constructors that can be used to create an object are as follows :

- String () without arguments create a String object with no character
- String (char ary[]) creates a String object with its initial value using ary argument
- String (char ary[], int start, int len) creates a String object using 1-D ary argument starting at ary[start] with len number of characters
- String (String strObj) creates a String object which is same as object specified in argument
- String (string literal) creates a String object that refers to the literal specified in argument.

Figure 9.15 shows the use of various types of String class constructors. Do not forget to use new operator while creating object.



```
/* creating String Objects */
class String1
{
        public static void main (String [] s)
        {
                char name [] = { 'J', 'a', 'v', 'a'};

                String str1 = new String();
                String str2 = new String(name);
                String str3 = new String(name, 1,2);
                String str4 = new String(str3);
                String str5 = new String("Java");

                System.out.println ("str1 is " + str1);
                System.out.println ("str2 is " + str2);
                System.out.println ("str3 is " + str3);
                System.out.println ("str4 is " + str4);
                System.out.println ("str5 is " + str5);
        } // main
} // class
```

```
>javac String1.java
>Exit code: 0
>java -cp . String1
str1 is
str2 is Java
str3 is av
str4 is av
str5 is Java
>Exit code: 0
```

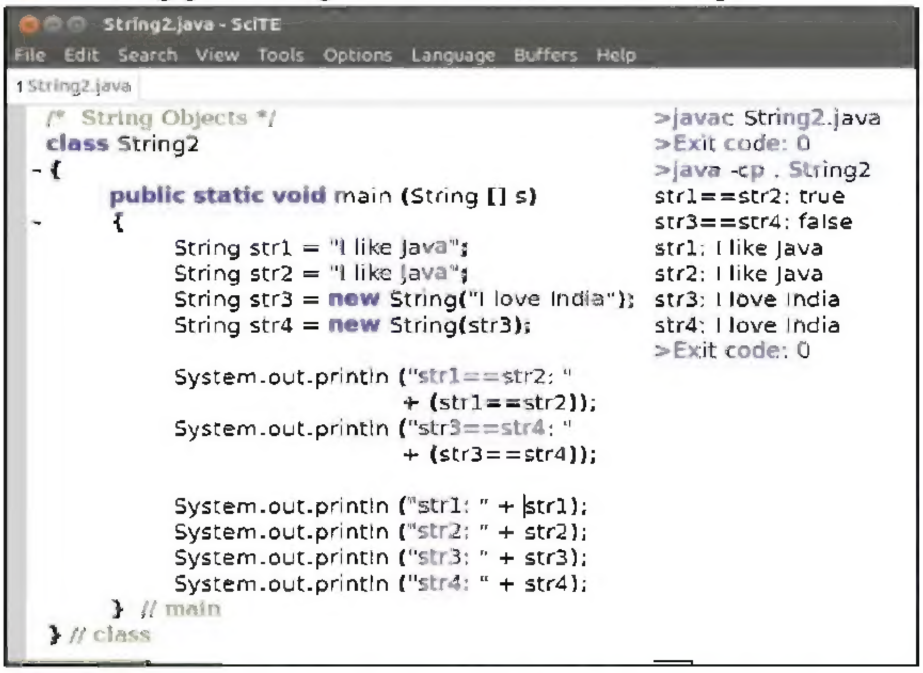
Figure 9.15 : Creating objects of class String using various constructors

In Java, characters are stored using two bytes. To save space, if the characters are ASCII, we should use an array of bytes instead of array of characters. We can use a constructor with array of bytes as an argument. Try the same program by replacing char array with byte array.

Note :

Literals are stored in memory. When two String objects are created using same string literals, memory space is not allocated for second object. Both objects refer to same memory location.

Separate memory is allocated when string objects are created using new operator even if strings are identical. Code listing given in figure 9.16 shows such an example.



```
/* String Objects */
class String2
{
        public static void main (String [] s)
        {
                String str1 = "I like Java";
                String str2 = "I like Java";
                String str3 = new String("I love India");
                String str4 = new String(str3);

                System.out.println ("str1==str2: "
                                    + (str1==str2));
                System.out.println ("str3==str4: "
                                    + (str3==str4));

                System.out.println ("str1: " + str1);
                System.out.println ("str2: " + str2);
                System.out.println ("str3: " + str3);
                System.out.println ("str4: " + str4);
        } // main
} // class
```

```
>javac String2.java
>Exit code: 0
>java -cp . String2
str1==str2: true
str3==str4: false
str1: I like Java
str2: I like Java
str3: I love India
str4: I love India
>Exit code: 0
```

Figure 9.16 : Identical String objects created using string literal and using new operator

In code listing given in figure 9.16, note that "str1 == str2" compares the contents of str1 and str2 and not the objects at str1 and str2. String objects str1 and str2 are created using same string literal but without using new operator. Here, both reference variables str1 and str2 refer to the same instance as created for str1. For object str2, separate memory is not allocated as seen in figure 9.17. In the same program, String objects str3 and str4 are created using new operator. The contents of both these objects are same and they refer to different memory locations. Separate memory is allocated for these two objects as seen in figure 9.17.

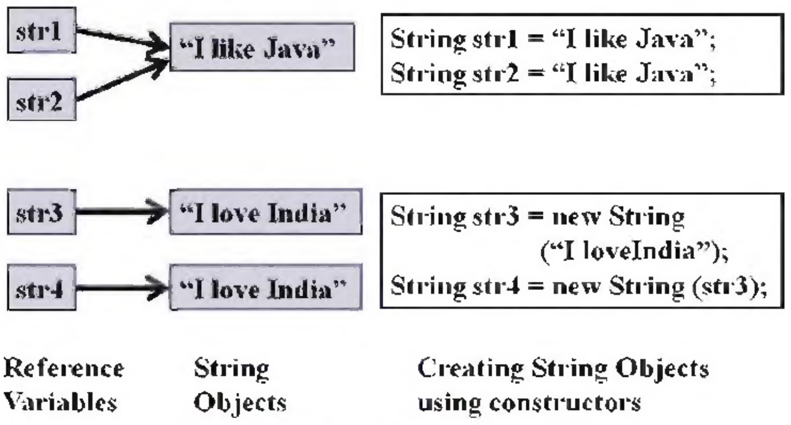
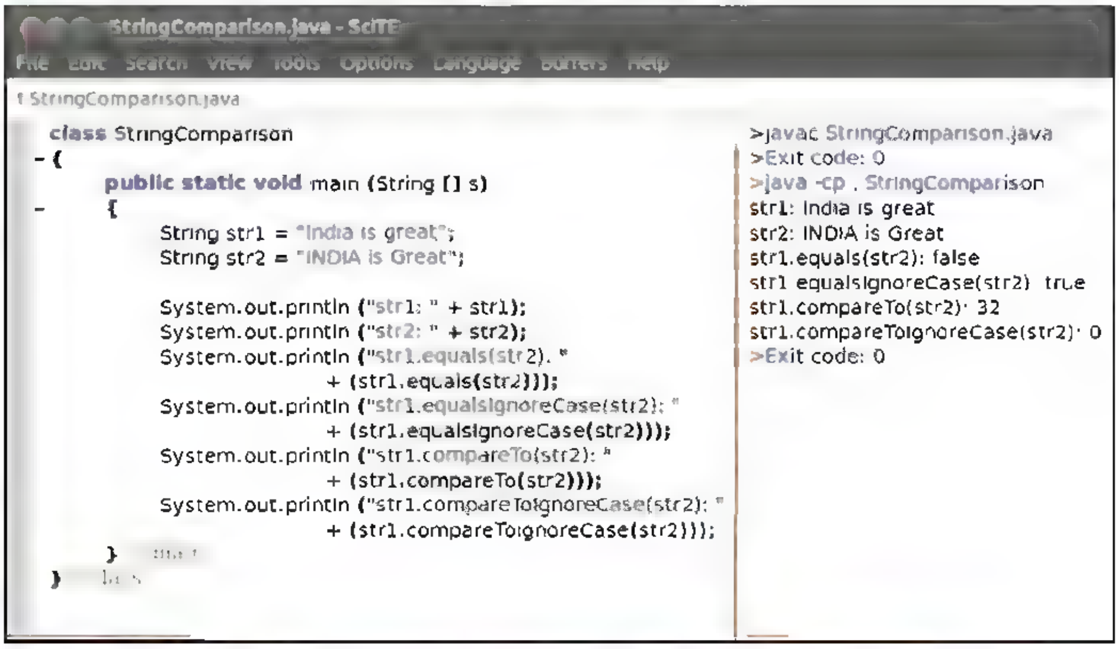

str1
str2
"I like Java"
String str1 = "I like Java";
String str2 = "I like Java";
str3
str4
"I love India"
"I love India"
String str3 = new String
("I loveIndia");
String str4 = new String (str3);
Reference Variables
String Objects
Creating String Objects using constructors

Figure 9.17: Identical strings

String class methods

The String class provides methods to compare strings, find length of string, combining strings, obtaining substrings, converting strings, splitting strings, searching for character or patterns in string etc. We will see the examples of using some of these methods through programs. The program in figure 9.18 shows different ways to compare strings or part of strings.


StringComparison.java - SciTE
File Edit Search View Tools Options Language Buffers Help
StringComparison.java

```
class StringComparison
{
        public static void main (String [] s)
        {
                String str1 = "India is great";
                String str2 = "INDIA is Great";

                System.out.println ("str1: " + str1);
                System.out.println ("str2: " + str2);
                System.out.println ("str1.equals(str2): "
                                + (str1.equals(str2)));
                System.out.println ("str1.equalsIgnoreCase(str2): "
                                + (str1.equalsIgnoreCase(str2)));
                System.out.println ("str1.compareTo(str2): "
                                + (str1.compareTo(str2)));
                System.out.println ("str1.compareToIgnoreCase(str2): "
                                + (str1.compareToIgnoreCase(str2)));
        }
}
```

```
>javac StringComparison.java
>Exit code: 0
>java -cp . StringComparison
str1: India is great
str2: INDIA is Great
str1.equals(str2): false
str1 equalsIgnoreCase(str2) true
str1.compareTo(str2): 32
str1.compareToIgnoreCase(str2): 0
>Exit code: 0
```

Figure 9.18 : Comparing strings using methods of class String

The syntax and description of various comparison methods is shown in table 9.1.

Method	Description
boolean equals (String str)	Returns true if invoking string is same as str
boolean equalsIgnoreCase (String str)	Returns true if invoking string is same as str after ignoring case (case insensitive)
int compareTo (String str)	Returns 0, >0, <0 integer if invoking string is equal to, greater than or less than str respectively
int compareToIgnoreCase (String str)	Same as CompareTo but case insensitive

Table 9.1 : String class methods for comparing strings

String class provides methods for other tasks as follows. Some of them are given in table 9.2.

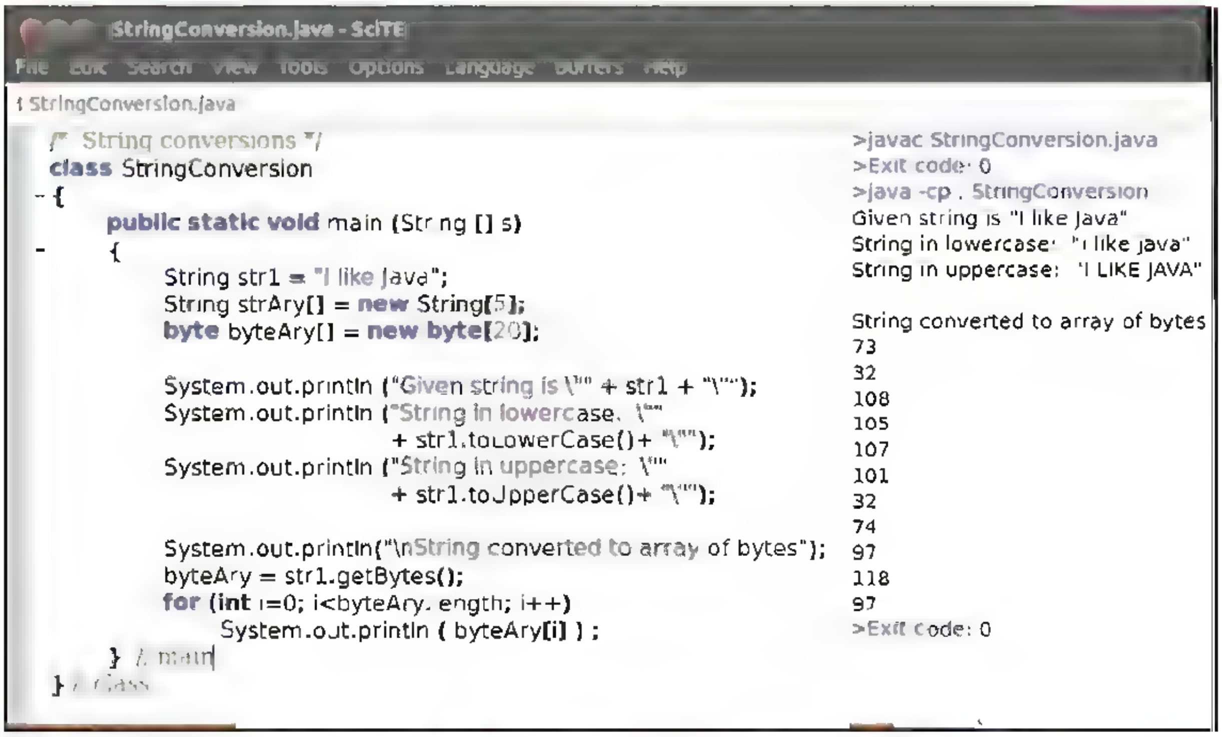
- Extracting part of string
- Replacing characters or substrings
- Splitting string into substrings
- Getting number of characters

- Getting character at specified index position
- Converting string into an array of bytes
- Converting string into lowercase or uppercase
- Appending string
- Copy string or part of string

Method	Description
int length()	Returns number of characters in invoking string
char indexAt(int index)	Returns character at index position from the invoking string, index considered from 0
byte [] getBytes()	Returns an array of characters as bytes from invoking string
void getChars (int fromIndx, int toIndx, char target [], int targetIndx)	Copies characters of invoking string from fromIndx to toIndx-1 to target array from targetIndx onwards
String concat(String str)	Returns a string after appending str with the invoking string
String toLowerCase()	Returns a string with all characters of invoking string converted to lowercase
String toUpperCase()	Returns a string with all characters of invoking string converted to uppercase

Table 9.2 : Additional methods of String class

The code listing given in figure 9.19 shows how to use some of these methods.


StringConversion.java - SciTE

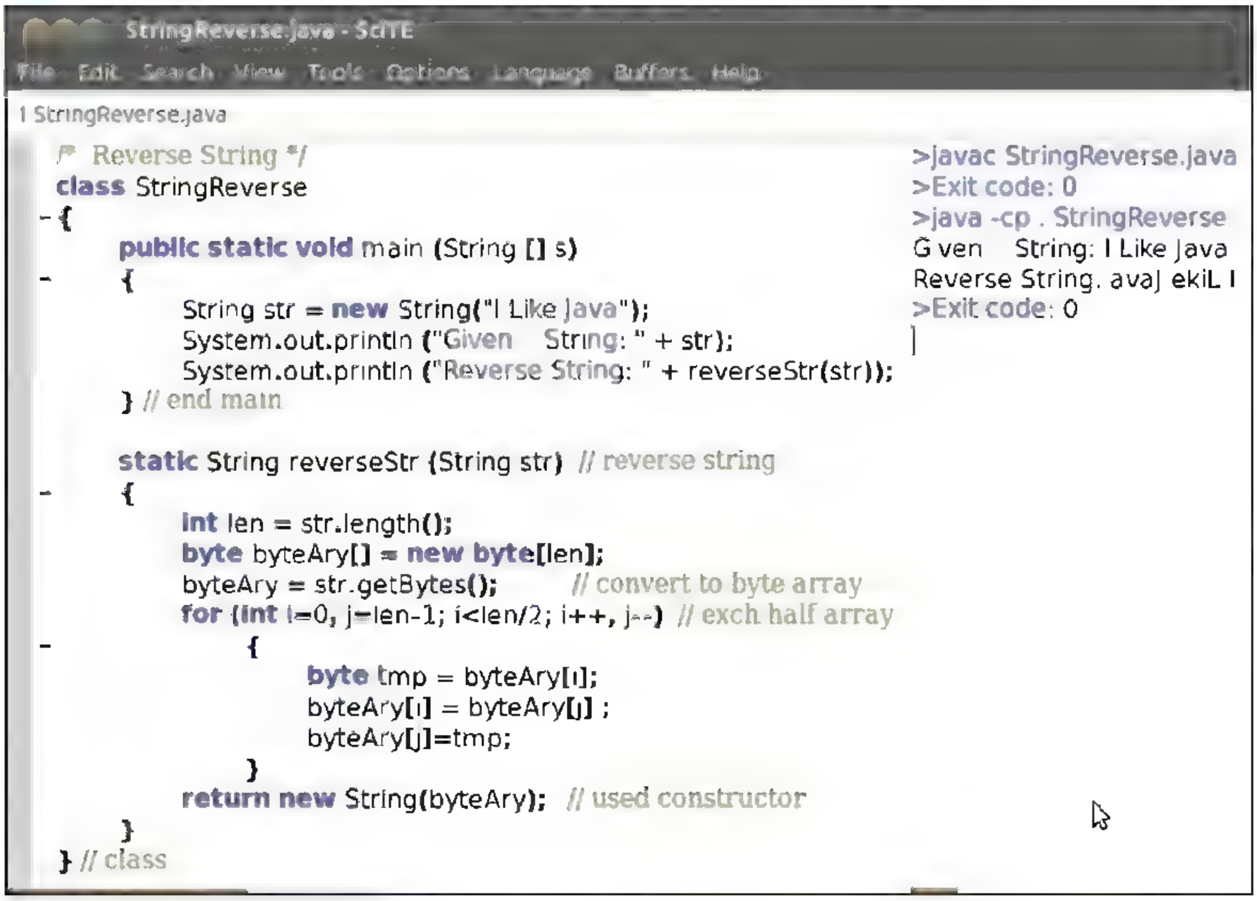
```
/* String conversions */
class StringConversion
{
    public static void main (String [] s)
    {
        String str1 = "I like Java";
        String strAry[] = new String[5];
        byte byteAry[] = new byte[20];

        System.out.println ("Given string is \"" + str1 + "\"");
        System.out.println ("String in lowercase: \""
                            + str1.toLowerCase()+ "\"");
        System.out.println ("String in uppercase: \""
                            + str1.toUpperCase()+ "\"");

        System.out.println("\nString converted to array of bytes");
        byteAry = str1.getBytes();
        for (int i=0; i<byteAry.length; i++)
            System.out.println ( byteAry[i] ) ;
    } // main
} // class
```

```
>javac StringConversion.java
>Exit code: 0
>java -cp . StringConversion
Given string is "I like Java"
String in lowercase: "i like java"
String in uppercase: "I LIKE JAVA"

String converted to array of bytes
73
32
108
105
107
101
32
74
97
118
97
>Exit code: 0
```

Figure 9.19 : Using String conversion methods

Note: With array variable, length is an attribute or property of array; whereas with String object, length is a method. So we need to use () while using length method with String object.

String class of Java does not provide any method to reverse a string. So, let us write a program to reverse string. Refer code listing and its output given in figure 9.20.


StringReverse.java - SciTE
File Edit Search View Tools Options Language Buffers Help
1 StringReverse.java

```java
/* Reverse String */
class StringReverse
{
        public static void main (String [] s)
        {
                String str = new String("I Like Java");
                System.out.println ("Given    String: " + str);
                System.out.println ("Reverse String: " + reverseStr(str));
        } // end main

        static String reverseStr (String str)  // reverse string
        {
                int len = str.length();
                byte byteAry[] = new byte[len];
                byteAry = str.getBytes();          // convert to byte array
                for (int i=0, j=len-1; i<len/2; i++, j--)  // exch half array
                {
                        byte tmp = byteAry[i];
                        byteAry[i] = byteAry[j] ;
                        byteAry[j]=tmp;
                }
                return new String(byteAry);   // used constructor
        }
} // class
```

```
>javac StringReverse.java
>Exit code: 0
>java -cp . StringReverse
Given    String: I Like Java
Reverse String: avaJ ekiL I
>Exit code: 0
```

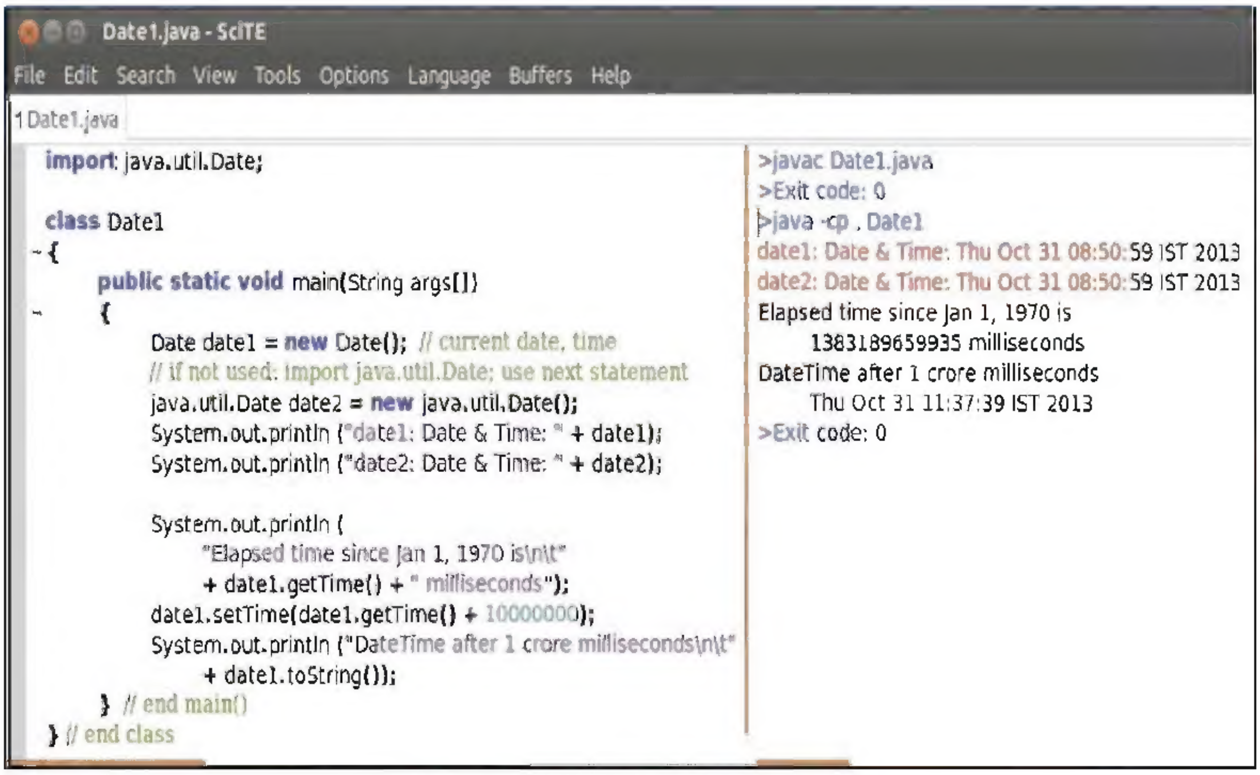
Figure 9.20 : Reversing a string

Here, we have written a user-defined method reverseStr that reverses the string as follows:

1. Determine the length of string using length() method of String class
2. Convert a string into an array of bytes using method getBytes() of String class
3. Exchange half of the elements on left (starting from first towards right) with half of the elements on right (starting from last towards left) of a byte array
4. Construct a String object from a byte array using constructor and return it

Date class

We have studied about String class and Arrays class provided by Java. Java library also provides Date class in java.util package. Date class encapsulate both date and time and represents the value using milliseconds precision. Figure 9.21 shows the use of Date class with code listing and its output.



```java
import java.util.Date;

class Date1
{
      public static void main(String args[])
      {
            Date date1 = new Date();  // current date, time
            // if not used: import java.util.Date; use next statement
            java.util.Date date2 = new java.util.Date();
            System.out.println ("date1: Date & Time: " + date1);
            System.out.println ("date2: Date & Time: " + date2);

            System.out.println (
                  "Elapsed time since Jan 1, 1970 is\n\t"
                  + date1.getTime() + " milliseconds");
            date1.setTime(date1.getTime() + 10000000);
            System.out.println ("DateTime after 1 crore milliseconds\n\t"
                  + date1.toString());
      } // end main()
} // end class
```

```
>javac Date1.java
>Exit code: 0
>java -cp . Date1
date1: Date & Time: Thu Oct 31 08:50:59 IST 2013
date2: Date & Time: Thu Oct 31 08:50:59 IST 2013
Elapsed time since Jan 1, 1970 is
        1383189659935 milliseconds
DateTime after 1 crore milliseconds
        Thu Oct 31 11:37:39 IST 2013
>Exit code: 0
```

Figure 9.21 : Using Date class

Table 9.3 lists some of the methods of Date class.

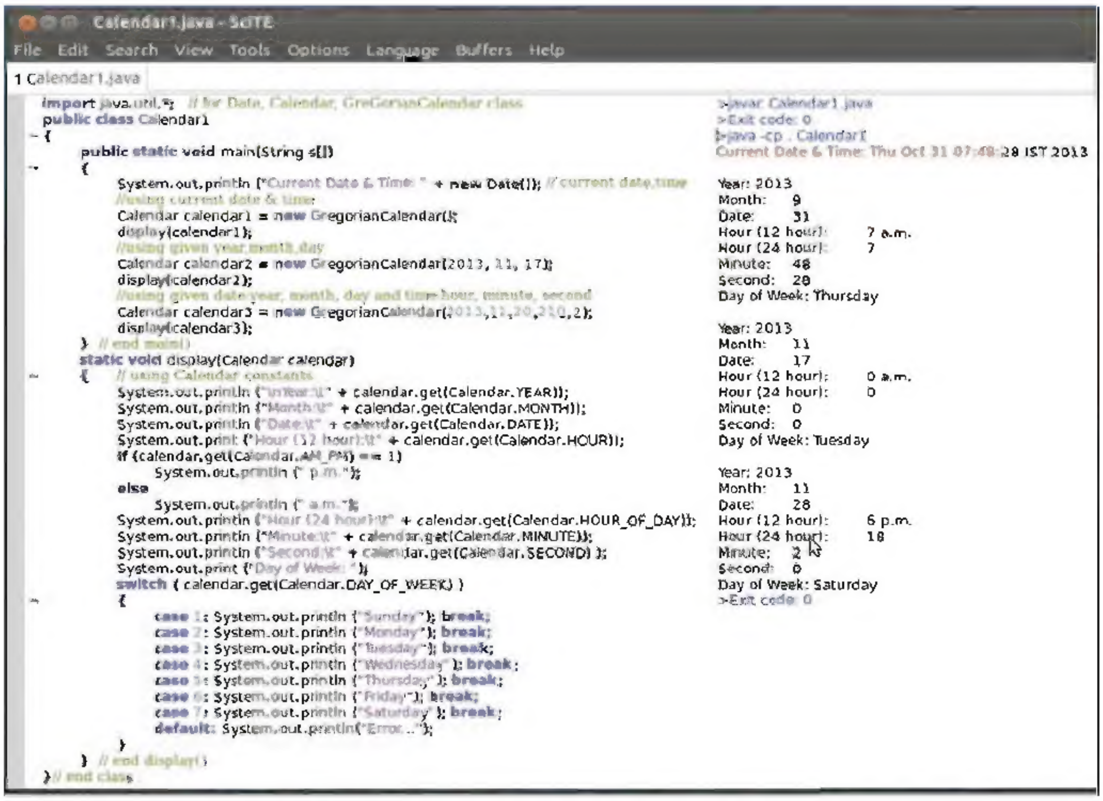
Method	Description
Date()	Constructs Date object using current system time
Date (long elapsedTime)	Constructs Date object using specified time in milliseconds elapsed since January 1, 1970 GMT
String toString()	Returns a string representing date and time of invoking object
long getTime()	Returns number of milliseconds since January 1, 1970 GMT
void setTime (long elapsedTime)	Sets new date and time of an object using elapsed time

Table 9.3 : Methods of Date class

Calendar class

Like Date class, Calendar class is also provided in java.util package. This class can be used to extract detailed calendar information like year, month, date, hour, minute and second. Here, we will see the use of GregorianCalendar subclass of Calendar class.

Figure 9.22 shows an example program to use Calendar class along with its output. Note the use of user-defined class method display() used to display various components of date and time.



```
import java.util.*;  // for Date, Calendar, GreGorianCalendar class
public class Calendar1
{
    public static void main(String s[])
    {
        System.out.println ("Current Date & Time: " + new Date()); // current date,time
        //using current date & time
        Calendar calendar1 = new GregorianCalendar();
        display(calendar1);
        //using given year,month,day
        Calendar calendar2 = new GregorianCalendar(2013, 11, 17);
        display(calendar2);
        //using given date-year, month, day and time-hour, minute, second
        Calendar calendar3 = new GregorianCalendar(2013,11,28,18,2);
        display(calendar3);
    } // end main()
    static void display(Calendar calendar)
    {   // using Calendar constants
        System.out.println ("\nYear:\t" + calendar.get(Calendar.YEAR));
        System.out.println ("Month:\t" + calendar.get(Calendar.MONTH));
        System.out.println ("Date:\t" + calendar.get(Calendar.DATE));
        System.out.print ("Hour (12 hour):\t" + calendar.get(Calendar.HOUR));
        if (calendar.get(Calendar.AM_PM) == 1)
            System.out.println (" p.m.");
        else
            System.out.println (" a.m.");
        System.out.println ("Hour (24 hour):\t" + calendar.get(Calendar.HOUR_OF_DAY));
        System.out.println ("Minute:\t" + calendar.get(Calendar.MINUTE));
        System.out.println ("Second:\t" + calendar.get(Calendar.SECOND) );
        System.out.print ("Day of Week: ");
        switch ( calendar.get(Calendar.DAY_OF_WEEK) )
        {
            case 1: System.out.println ("Sunday"); break;
            case 2: System.out.println ("Monday"); break;
            case 3: System.out.println ("Tuesday"); break;
            case 4: System.out.println ("Wednesday" ); break;
            case 5: System.out.println ("Thursday" ); break;
            case 6: System.out.println ("Friday"); break;
            case 7: System.out.println ("Saturday" ); break;
            default: System.out.println("Error...");
        }
    } // end display()
}// end class
```

```
>javac Calendar1.java
>Exit code: 0
>java -cp . Calendar1
Current Date & Time: Thu Oct 31 07:48:28 IST 2013

Year: 2013
Month:    9
Date:     31
Hour (12 hour):     7 a.m.
Hour (24 hour):     7
Minute:   48
Second:   28
Day of Week: Thursday

Year: 2013
Month:    11
Date:     17
Hour (12 hour):     0 a.m.
Hour (24 hour):     0
Minute:   0
Second:   0
Day of Week: Tuesday

Year: 2013
Month:    11
Date:     28
Hour (12 hour):     6 p.m.
Hour (24 hour):     18
Minute:   2
Second:   0
Day of Week: Saturday
>Exit code: 0
```

Figure 9.22 : using Calendar class and its constants

Like get methods, we can use set methods to set the value of the field constants of Calendar class. For example, if calendar is an object of class Calendar, then execution of call 'calendar.set(Calendar.DATE, 20)' will set the date to 20.

Table 9.4 lists the integer constants defined in Calendar class. These constants are given meaningful and self-explanatory names.

Constant	Description
YEAR	Year of calendar
MONTH	Month of calendar (0 for January, 11 for December)
DATE	Day of calendar month
DAY_OF_MONTH	Same as DATE
HOUR	Hour in 12-hour notation
HOUR_OF_DAY	Hour in 24-hour notation
MINUTE	Minute
SECOND	Second
AM_PM	0 for AM, 1 for PM
DAY_OF_WEEK	Day number within a week (1 for Sunday, 7 for Saturday)
WEEK_OF_MONTH	Week number within the month
WEEK_OF_YEAR	Week number within the year
DAY_OF_YEAR	Day number in the year (1 for the first day)

Table 9.4: Constants defined in Calendar class

Summary

This chapter explains how to deal with array, string and date. Array is used to have collection of variables of same data type. Java basically supports only 1-D arrays. Using 1-D array of arrays, we can practically get multi-dimensional arrays. Array is treated as an object of class Arrays, so array name is a reference variable referring to memory location where its elements are stored. All methods of class Arrays can be used with array objects. Examples of such methods are: sort, fill. String class provided in Java enables to work with a sequence of characters. Examples of operations that can be performed on strings are: comparing strings, finding number of characters in a string, converting case of letters of string. Class Date and Calendar are provided to work with date and time. Using Calendar class methods, we can get and set the fields like year, month, date, hour, minute, seconds.

EXERCISE

1. Explain array giving example.
2. Differentiate between 1-D and 2-D arrays.
3. Explain the use of the following classes :

   a. String b. Date c. Calendar

4. Choose the most appropriate from those given below :

   (1) Which of the following refer to the starting index value in arrays ?

   (a) 0 (b) 1 (c) null (d) All of these

   (2) What is the size of second dimension in an array sales[5][12] ?

   (a) 5 (b) 12 (c) 60 (d) 10

   (3) What will expression sales.length return for an array sales[5][12] ?

   (a) 5 (b) 12 (c) 60 (d) 120

   (4) When an array sales [5][12] is declared without specifying initial values, what is the initial value of sales[0][0] ?

   (a) 0 (b) default value (c) compilation error (d) 60

   (5) What does 'length' refer to for an object of String class ?

   (a) attribute (b) method (c) class variable (d) class name

   (6) If 'str' is the object of String class and its content is "Thank GOD", then what is the value of str.length() ?

   (a) 9 (b) 10 (c) 8 (d) 11

(7) What type of value is returned when we use get method of Calendar class with constant DAY_OF_WEEK as an argument ?

(a) int (b) char (c) String (d) boolean

LABORATORY EXERCISE

1. Write a program that uses an array to store 12 values of hourly temperature of a day from 6 am to 6 pm. Use either initialization or assignment statements to assign these values. Print maximum and minimum temperature of the day.
2. Write a program that stores weekly sale of five salespersons. A sales person is given a weekly incentive depending upon his/her average weekly sale. Incentive is: 10 % if average weekly sale is up to Rs. 10000, 15% if average weekly sale is above Rs.10000 but less than or equal to Rs.30000 and 20% if average weekly sale is above Rs. 30000. Compute total daily sale of all salespersons combined together. Also compute weekly incentive for each salesperson.
3. Write a program to see whether a string is palindrome or not. (String is palindrome if its reverse is same as original, example "1771", "madam")
4. Write a program to print current date in the format "DD-MMM-YYYY". For example, 17th November 2013 should be printed as 17-Nov-2013. Also print the day of week in words.
5. Write a program to set the date and time as per your birth date and time. Print the week day on your birthday.

●